Mobile development
Assignment 2: Create Basic Pages of Instagram in Android Studio
Bitanov Assanali (23MD0392)
18.10.2024

Table of Contents

INTRODUCTION

In this assignment, we are going to practice our development skills in order to build an app with basic Instagram pages and a navigation panel. The app is built with Jacket Compose. The topics that are covered within the project are layouts, state management, LazyColumn, navigation, Material3, and preview. The pages that we need to build are Home Feed page, Profile page, Search page, Add post page, and Notifications page.

Project Setup

First, we create a new directory for the project. The name of the project is Assignment2. In this folder, we create the MainActivity.kt file. The MainActivity.kt file is a Kotlin source code file typically used in Android app development. This file contains the main activity of an Android app, serving as the entry point for the application [1]. The file is shown in Figure 1.

```kotlin
package com.example.assginment2

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.navigation.compose.rememberNavController
import com.example.assginment2.ui.navigation.BottomNavItem
import com.example.assginment2.ui.navigation.BottomNavigationBar
import com.example.assginment2.ui.navigation.NavigationGraph
import com.example.assginment2.ui.theme.Assginment2Theme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Assginment2Theme {
                MainScreen()
            }
        }
    }
}
```

Figure 1. The MainActivity.kt file

We use “Assginment2Theme” theme in the app. The theme file was automatically created with an empty activity project. The project is made with the Jacket Compose toolkit. It is an intuitive yet powerful modern tool that requires less code, comparing to the XML-using approach [2].

Home Feed Page

We define two composable functions in the HomeFeedScreen.kt file. HomeFeedScreen and PostItem are part of the user interface for displaying a list of posts in the home feed of the application. The code of this file is shown in Figure 2.

```
@Composable
fun HomeFeedScreen() {
    LazyColumn(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        items(PostRepository.posts.size) { index ->
            PostItem(postIndex = index)
        }
    }
}

@Composable
fun PostItem(postIndex: Int) {
    val post = PostRepository.posts[postIndex]

    Column(modifier = Modifier.padding(bottom = 16.dp)) {
        Text(text = post.username, modifier = Modifier.padding(bottom = 8.dp))
        Image(
            painter = painterResource(id = post.imageResId),
            contentDescription = post.caption,
            modifier = Modifier
                .fillMaxWidth()
                .height(200.dp),
            contentScale = ContentScale.Crop
        )
        Text(text = post.caption, modifier = Modifier.padding(top = 8.dp))
        Text(text = "${post.likes} likes", modifier = Modifier.padding(top = 4.dp))
    }
}
```

Figure 2. The HomeFeedScreen.kt file

The HomeFeedScreen creates a scrollable list using LazyColumn [3]. The PostItem function displays every post. The post data is stored in the PostRepository.kt file. The file is shown in Figure 3.

```kotlin
data class Post(
    val id: Int,
    val username: String,
    val caption: String,
    val imageResId: Int,
    val likes: Int
)

object PostRepository {
    private var currentId = 0

    // List to store posts
    private val _posts = mutableStateListOf(
        Post(
            id = currentId++,
            username = "asan123",
            caption = "My first post!",
            imageResId = R.drawable.sample_image1,
            likes = 120
        ),
        Post(
            id = currentId++,
            username = "asan123",
            caption = "My second post!",
            imageResId = R.drawable.sample_image5,
            likes = 120
```

```kotlin
val posts: List<Post>
    get() = _posts

fun addPost(post: Post) {
    _posts.add(post)
}

fun nextId(): Int {
    return currentId++
}
```

Figure 3. The PostRepository.kt file

We define Post class and PostRepository object. In the class we define parameters id, username, caption, imageResId and likes. In the repository we define the users.

The Home Feed page displays all posts, users, captions and likes. The page is shown in Figure 4.

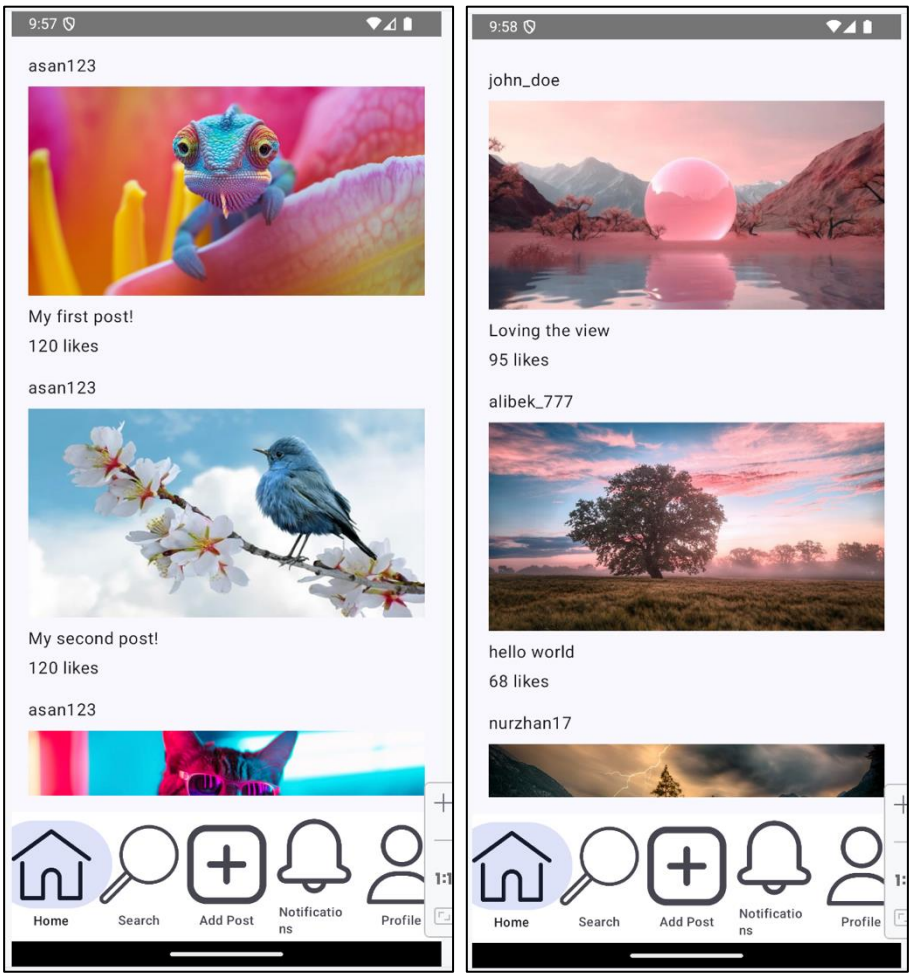

Figure 4. The Home Feed page

There we can see several posts with images, captions, and likes. When users upload a post they will be displayed here. The Home Feed page shows all the uploaded posts by the users.

Profile Page

The Profile page will include user's profile picture, username, bio, number of posts, and the posts themselves. We define the logic in ProfileScreen.kt function. The code of the function is shown in Figure 5.

```
@Composable
fun ProfilePage(username: String) {
    val user = UserRepository.users.find { it.username == username }
    val userPosts = PostRepository.posts.filter { it.username == username }

    Column(modifier = Modifier.fillMaxSize().padding(16.dp)) {
        user?.let {
            Row(modifier = Modifier.fillMaxWidth().padding(bottom = 16.dp)) {
                Image(
                    painter = painterResource(id = it.profileImageResId),
                    contentDescription = "${it.username}'s Profile Picture",
                    modifier = Modifier
                        .size(80.dp)
                        .padding(end = 16.dp),
                    contentScale = ContentScale.Crop
                )
                Column {
                    Text(text = it.username)
                    Text(text = it.bio, modifier = Modifier.padding(top = 4.dp))
                    Text(text = "Posts: ${userPosts.size}", modifier = Modifier.padding(top = 4.dp))
                }
            }

            LazyVerticalGrid(
                columns = GridCells.Fixed( count: 3),
                modifier = Modifier.fillMaxHeight()
            ) {
                items(userPosts.size) { index ->
                    PostGridItem(post = userPosts[index])
                }
            }
```

Figure 5. The code of the ProfileScreen.kt file

We create a composable function ProfilePage where we retrieve user data from UserRepository. LazyVerticalGrid creates a scrollable grid of users's posts [4]. In this case, the user have 3 posts. The UserRepository.kt file is shown in Figure 6.

```kotlin
data class User(
    val username: String,
    val bio: String,
    val profileImageResId: Int
)

object UserRepository {
    val users = listOf(
        User(
            username = "asan123",
            bio = "Software Developer",
            profileImageResId = R.drawable.sample_image1
        ),
        User(
            username = "john_doe",
            bio = "Photographer and Traveler",
            profileImageResId = R.drawable.sample_image2
        ),
```

Figure 6.The UserRepository.kt file

The ProfilePage is designed to provide a visually structured and user-friendly interface that displays key information about the user at the top, followed by a grid layout showing their posts. The user's profile page is shown in Figure 7.

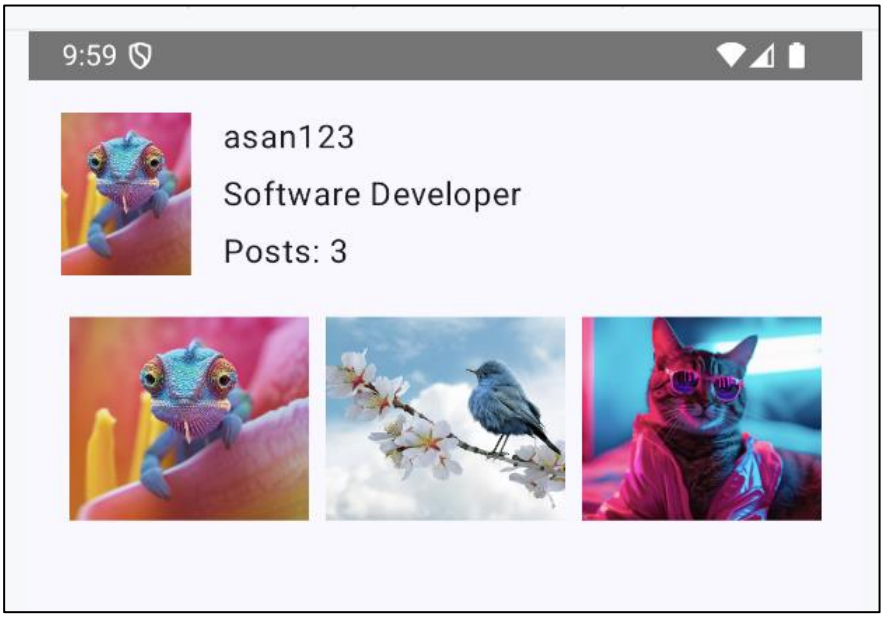

Figure 7. asan123's profile page

Search Page

The Search Page contains search bar at the top and a list of the users. When a user types a username, they can find that user. The logic is defined the SearchScreen.kt file. The code if this file is shown in Figure 8.

```
@Composable
fun SearchPage() {
    var query by remember { mutableStateOf( value: "") }
    val filteredUsers = UserRepository.users.filter { it.username.contains(query, ignoreCase = true) }

    Column(modifier = Modifier
        .fillMaxSize()
        .padding(16.dp)) {

        SearchBar(query, onQueryChanged = { query = it })

        Spacer(modifier = Modifier.height(16.dp))

        if (filteredUsers.isEmpty()) {
            Text(text = "No users found", modifier = Modifier.padding(16.dp))
        } else {
            LazyColumn {
                items(filteredUsers.size) { index ->
                    SearchResultItem(userIndex = index, filteredUsers)
                }
            }
        }
    }
}
```

Figure 8. The code of the SearchScreen.kt file

The query var stores the user's search input, and this state is updated as the user types into the search bar. Then, we find that user by their username in UserRepository. The result is shown in Figure 9.

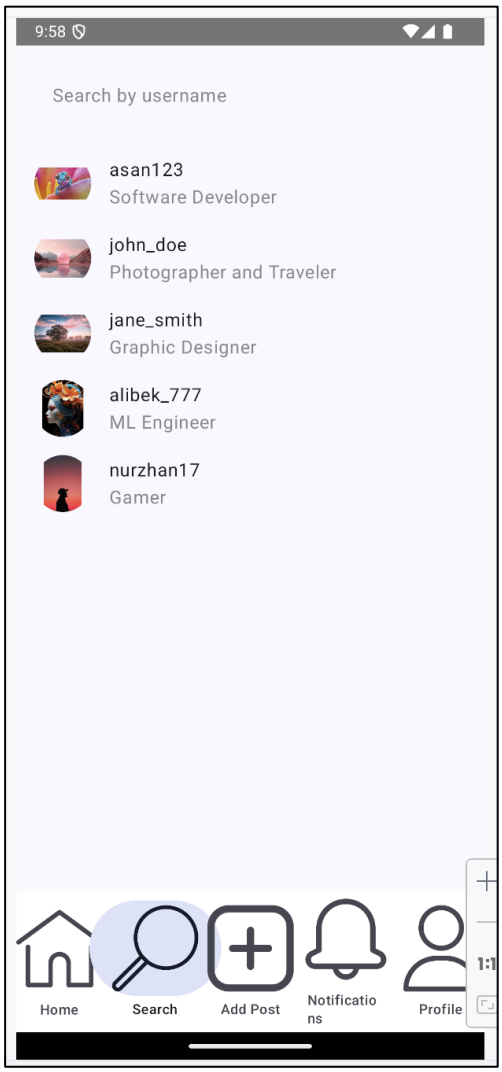

Figure 9. The Search Screen page

There we have five uesrs. They are listed by creation date. At the top, we have the search bar. When we begin to type some username we can see the user. This process is shown in Figure 10.

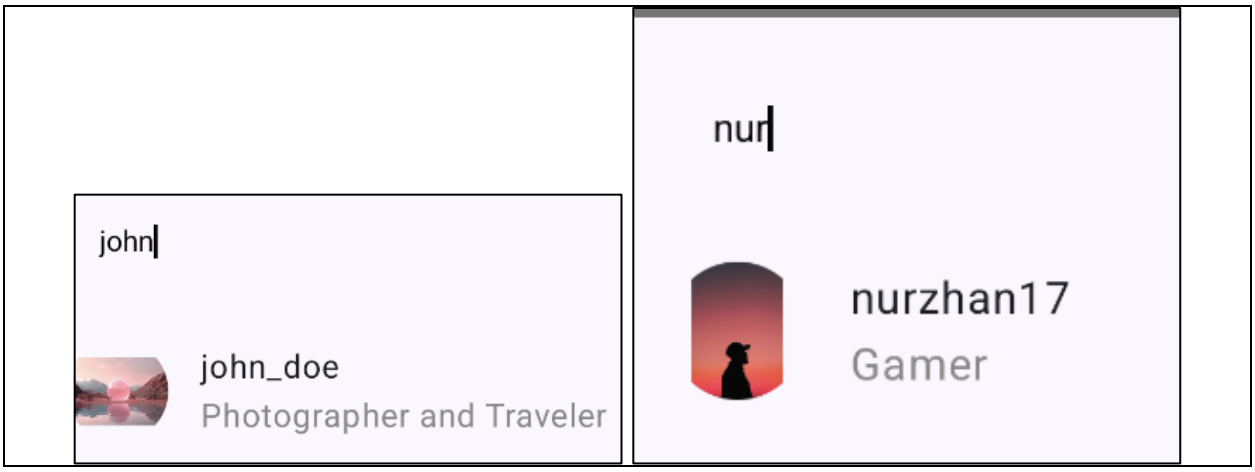

Figure 10. Searching for some users

As shown in the Figure, we can see the user just by typing the letter that contains their usernames. It's an efficient way to find a user as we don't have to write or even know the full username. As we type a username, we can see their profile picture, full username, and bio. The design itself is simple yet efficient, this helps users find each other more easily and operative.

Add post Page

In the Add Post Page we define sample image, text field for caption, and upload button. The logic of the page is stored in AddPostScreen.kt file. The file is shown in Figure 11.

```
@Composable
fun AddPostScreen(onPostAdded: () -> Unit) {
    var caption by remember { mutableStateOf( value: "") }
    var selectedImage by remember { mutableStateOf(R.drawable.sample_image1) }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Image(
            painter = painterResource(id = selectedImage),
            contentDescription = "Selected Image",
            modifier = Modifier
                .fillMaxWidth()
                .height(200.dp),
            contentScale = ContentScale.Crop
        )

        Spacer(modifier = Modifier.height(16.dp))

        BasicTextField(
            value = caption,
            onValueChange = { caption = it },
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 16.dp)
        )
        Button(
            onClick = {
                val newPost = Post(
                    id = PostRepository.nextId(),
                    username = "CurrentUser",
                    caption = caption,
                    imageResId = selectedImage,
                    likes = 0
                )
                PostRepository.addPost(newPost)
                onPostAdded()
            },
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(text = "Upload Post")
        }
    }
}
```

Figure 11. The AddPostScreen.kt file

We create a composable function AddPostScreen where we define all the logic of the page. The function uses mutable state variables to manage the post's caption in var caption and the selected image in var selectedImage. The user can input a caption using a BasicTextField, while the selected image is displayed at the top of the screen.

When the user uploads the post, it will be displayed on the Home Feed Page and the user's profile page. The Add Post Page is shown in Figure 12.

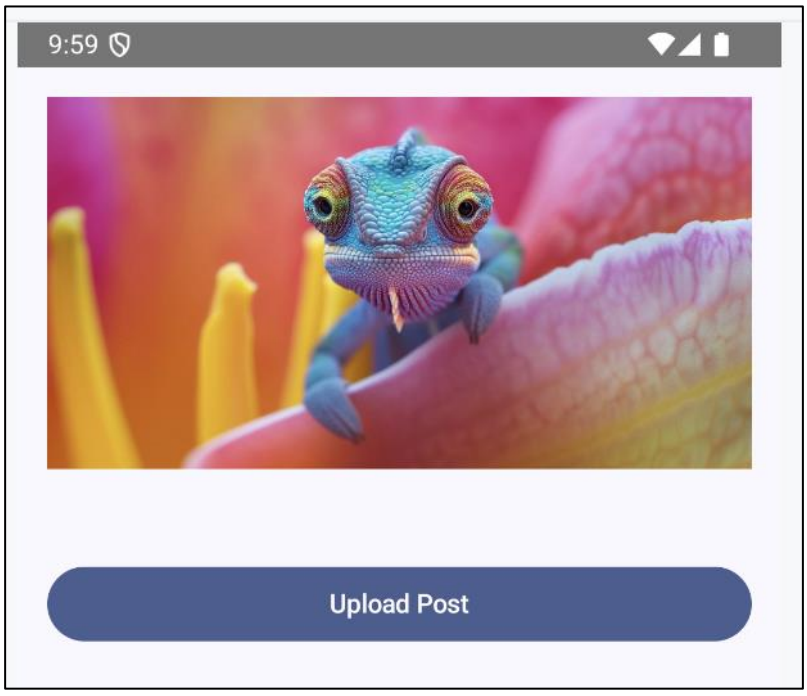

Figure 12. The Add Post Page

The interface is structured within a Column, which displayes the image, caption input, and button vertically. This layout is simple and clear for users to interact with. The image is shown at the top, filling the entire width of the screen. It's shown in a way that keeps the right proportions, so it looks nice and fits well with the rest of the design.

Notifications Page

In the notification page we display the notification that contains likes and comments. The code for this page is located in NotificationsScreen.kt file. Firstly, we create Notification class and NotificationRepository object. This piece of code is shown in Figure 13.

```
data class Notification(
    val type: String,
    val message: String
)

object NotificationRepository {
    val notifications = listOf(
        Notification(type = "like", message = "asan123 liked your post"),
        Notification(type = "comment", message = "john_doe commented: Nice photo!"),
        Notification(type = "like", message = "jane_smith liked your post"),
        Notification(type = "comment", message = "asan123 commented: Great work!"),
        Notification(type = "like", message = "john_doe liked your post")
    )
}
```

Figure 13. Notification class and NotificationRepository object

Next, we define the NotificationScreen function. The function displays a list of notifications using a vertical scrollable column LazyColumn. It loads all the notifications from the NotificationRepository and shows them on the screen.

```
@Composable
fun NotificationScreen() {
    LazyColumn(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        items(NotificationRepository.notifications.size) { index ->
            NotificationItem(notification = NotificationRepository.notifications[index])
        }
    }
}

@Composable
fun NotificationItem(notification: Notification) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 8.dp)
    ) {
        Text(text = notification.message)
    }
}
```

Figure 14. The NotificationScreen function

The screen fills the entire available space and includes padding to prevent elements from being too close to the edges. The Add Post page is shown in Figure 15.

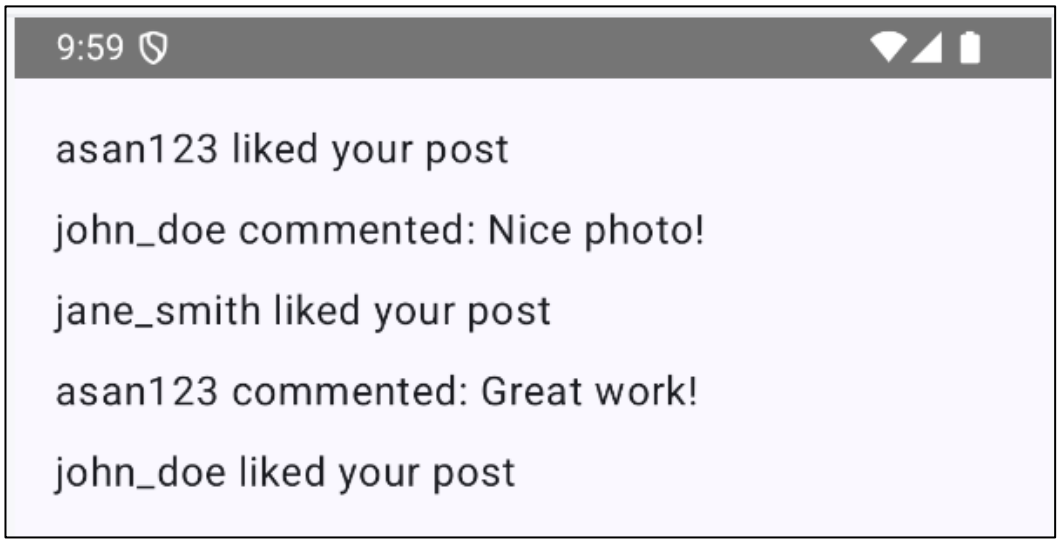

Figure 15. The Add Post Page

Here we can see the notifications. Each notification is displayed as text within a Column, with a small gap between them to provide good readability.

The Navigation Bar

The navigation bar is located at the bottom of each page. The bar allows to switch between the Home Feed, Search, Profile, Add Post, and Notifications pages. Each bottom has its icon. In order to implement the Navigation bar we created a new directory "navigation", where we store these files: BottomNavigationBar.kt, BottomNavItem.kt, and NavigationGraph.kt. The code of the BottomNavigationBar is shown in Figure 16.

```
@Composable
fun BottomNavigationBar(navController: NavController, items: List<BottomNavItem>) {
    NavigationBar(
        containerColor = Color.White,
        tonalElevation = 8.dp
    ) {
        val navBackStackEntry by navController.currentBackStackEntryAsState()
        val currentRoute = navBackStackEntry?.destination?.route

        items.forEach { item ->
            NavigationBarItem(
                icon = {
                    Icon(
                        painter = painterResource(id = item.icon),
                        contentDescription = item.title
                    )
                },
                label = { Text(text = item.title) },
                selected = currentRoute == item.route,
                onClick = {
                    if (currentRoute != item.route) {
                        navController.navigate(item.route) {
                            popUpTo(navController.graph.startDestinationId) { saveState = true }
                            launchSingleTop = true
                            restoreState = true
                        }
```

Figure 16. The BottomNavigationBar.kt file

The BottomNavigationBar takes a list of items from the BottomNavItem.kt file, each representing a screen, and a NavController to manage navigation. The current screen is determined by tracking the navBackStackEntry and comparing it with the item's route. When a user taps an item, the app navigates to the corresponding screen, ensuring that the navigation stack is managed efficiently with popUpTo, launchSingleTop, and restoreState.

We define the BottomNavItem class in the BottomNavItem.kt file. There we create these objects: Home, Search, AddPost, Notifications and Profile. The class is shown in Figure 17.

```
sealed class BottomNavItem(val title: String, val route: String, val icon: Int) {
    object Home : BottomNavItem( title: "Home",  route: "home", R.drawable.ic_home)
    object Search : BottomNavItem( title: "Search",  route: "search", R.drawable.ic_search)
    object AddPost : BottomNavItem( title: "Add Post",  route: "add_post", R.drawable.ic_add_post)
    object Notifications : BottomNavItem( title: "Notifications",  route: "notifications", R.drawable.ic_notifications)
    object Profile : BottomNavItem( title: "Profile",  route: "profile", R.drawable.ic_profile)
}
```

Figure 17. The BottomNavItem class

Each object has these three properties: title - the label displayed in the navigation bar, route - the navigation path, and icon - the resource ID for the item's icon.

In the NavigationGraph.kt file manages how different screens are linked and navigated based on the bottom navigation items. The file is shown in Figure 18.

```
@Composable
fun NavigationGraph(navController: NavHostController) {
    NavHost(
        navController = navController,
        startDestination = BottomNavItem.Home.route
    ) {
        composable(BottomNavItem.Home.route) {
            HomeFeedScreen()
        }
        composable(BottomNavItem.Search.route) {
            SearchPage()
        }
        composable(BottomNavItem.AddPost.route) {
            AddPostScreen(onPostAdded = {})
        }
        composable(BottomNavItem.Notifications.route) {
            NotificationScreen()
        }
        composable(BottomNavItem.Profile.route) {
            ProfilePage(username = "asan123")
        }
    }
}
```

Figure 18. The NavigationGraph.kt file

The NavHost is the container for managing the navigation, and the startDestination is set to the Home screen route BottomNavItem.Home.route, which is the first screen shown when the app starts.

Conclusion

In this assignment, we practiced creating an Android application. The Jetpack Compose was chosen as the main toolkit. We created a simple social media app. To complete this project we used various tools like: LazyColumns, NavigationBar, Compose, Preview, and Material3. Also, we enhanced our experience with Kotling language and Android Studio. And the structure of the project was a useful experience as we improved our knowledge about Android Apps structure.

References

1. MainActivity.kt [REINTECH] https://reintech.io/term/mainactivity-file-android-development
2. XML vs. Jetpack Compose: A Comparison [Open Replay] https://blog.openreplay.com/xml-vs-jetpack-compose--a-comparison/#:~:text=XML%20describes%20the%20format%20and,accelerates%20UI%20development%20on%20Android.
3. Lists LazyColumn and LazyRow [Metanit] https://metanit.com/kotlin/jetpack/2.6.php
4. LazyVerticalGrid [Jetpack Compose Playground] https://foso.github.io/Jetpack-Compose-Playground/foundation/lazyverticalgrid/